Empirical evaluation of the conjunct use of MOF and OCL

Juan Cadavid[1], Benoit Baudry[1], Benoît Combemale[2]

[1]INRIA, Centre Rennes – Bretagne Atlantique
Campus de Beaulieu, 35042 Rennes Cedex, France
{juan.cadavid,benoit.baudry}@inria.fr
[2]IRISA, Université de Rennes 1 Triskell Team, Rennes, France
{benoit.combemale}@irisa.fr

Abstract. MOF and OCL are commonly used for metamodeling: MOF to model the domain structure, and OCL for the well-formedness rules. Thus, metamodelers have to master both formalisms and understand how to articulate them in order to build metamodels that accurately capture domain knowledge. A systematic empirical analysis of the conjunct use of MOF and OCL in existing metamodels could help metamodelers understand how to use these formalisms. However, existing metamodels usually present anomalies that prevent automatic analysis without prior fixing. In particular, it often happens that both parts of the metamodel (MOF and OCL) are inconsistent. In this paper, we propose a process for analyzing metamodels and we report on the pre-processing phase we went through on 52 metamodels in order to get them ready for automatic empirical analysis.

1 Introduction

The Meta-Object Facility (MOF) [2] and the Object Constraint Language (OCL) [4] are commonly used for metamodeling: MOF to define a domain model and OCL to define well-formedness rules in this domain. At first glance, the roles of both standards seem well-delimited, yet many conceptual decisions can be implemented in either one, as we will demonstrate. For this reason, their combined usage has revealed several styles as observed in the panorama of developed mature DSMLs (Domain Specific Modeling Languages). Systematic empirical analysis of how these standards are combined in publicly available metamodels would help understand their usage and propose new methodologies and techniques to assist domain experts when building a new metamodel. Empiric analysis can be established through the systematic collection of metrics over existing metamodels. However, there currently exists no metrics that relate MOF and OCL and there exists no tool to automatically compute metrics on metamodels. Another issue is related to the lack of homogeneous formats to support the automatic analysis of MOF/OCL based metamodels. For example, OCL well-formedness rules are provided in various formats (txt, annotations in Ecore, OCL model). Also, because MOF and OCL parts are not always stored in the same format, both parts

of the metamodel tend to be inconsistent. When gathering data for empirical analysis, it is thus necessary to fix it prior to metrics computation. This paper aims at illustrating the challenges of the conjunct usage of MOF and OCL for metamodeling. We propose initial metric definitions and analysis methodology to empirically understand how both formalisms are related and conjunctly used in existing metamodels. We have collected 52 metamodels for which we have learned a few initial lessons by manually analyzing and fixing them in order to get them ready for automatic measurement. In particular we have found and fixed a number of inconsistencies in OCL invariants defined in OMG (Object Management Group) standard metamodels. Section 2 presents the motivation for this study. Section 3 introduces the problem statement, introduces definitions as well as the presentation of the two standards. Section 4 presents our research process. Section 5 and 6 present the first data findings and conclusions relevant to the first phase of our research process.

2 Motivation

This section illustrates the issues for the definition of a correct and precise metamodel through an example. The model in figure 1, expressed in the basic version of MOF called EMOF (Essential MOF), specifies the concepts and relationships of the Petri net domain. A `PetriNet` is composed of several `Arc`s and several `Node`s. `Arc`s have a source and a target `Node`, while `Node`s can have several incoming and outgoing `Arc`s. The model distinguishes between two different types of `Node`s: `Place`s or `Transition`s.

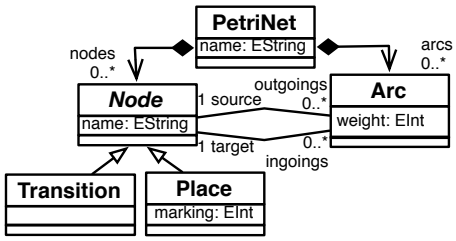

Fig. 1. MOF-based domain structure for Petri nets

This model captures every necessary concept to build Petri nets. However, there can also exist valid instances of this model that are not valid Petri nets. For example, the model does not prevent the construction of a Petri net in which an arc's source and target are only places, instead of linking a place and a transition. Thus, the sole model is not sufficient to precisely model the specific domain of Petri nets, since it still allows the construction of models that are not valid in this domain. The model needs to be enhanced with additional properties to capture the domain more precisely. The following well-formedness rules, expressed in OCL, show some mandatory properties of Petri nets.

1. *noEqualNamesInv*: Two nodes cannot have the same name.

```
context PetriNet inv :
self.nodes->forAll(n1, n2 | n1 <> n2
implies n1.name <> n2.name)
```

2. *noSameEndTypesInv*: No arc may connect two places or two transitions.

```
context Arc inv: self.source.oclType()
<> self.target.oclType()
```

3. *placeMarkingPositiveInv*: A place's marking must be positive.

```
context Place inv: self.marking >= 0
```

4. *arcWeightPositiveInv*: An arc's weight must be strictly positive.

```
context Arc inv: self.weight > 0
```

In our study we consider that the metamodel for Petri nets is the composition of the model and the associated well-formedness rules. We learn from this example that the construction of a precise metamodel, that consistently captures a domain, requires: (i) mastering two formalisms[1]: EMOF for concepts and relationships; OCL for properties; (ii) building two complimentary views on the domain model; (iii) finding a balance between what is expressed in one or the other formalism, (iv) keeping the views, expressed in different formalisms, consistent. This last point is particularly challenging in case of evolution of one or the other view. One notable case from the OMG and the evolution of the UML standard is that the `AssociationEnd` class disappeared after version 1.4 in 2003, but as late as version 2.2, released in 2009, there were still OCL expressions referring to this metaclass [11]. In the same manner, the OCL 2.2 specification depends on MOF 2.0, however a particular section of the specification defining the binding between MOF and OCL [4, p.169] makes use of the class `ModelElement` which only existed until MOF 1.4.

3 Research problem

3.1 Definitions

This section defines the terms we use to designate the focus of our analysis. The relationship between a model and a metamodel can be described as shown in figure 2 [3]. Here the `conformsTo` relation is a predicate function that returns true if all objects in the model are instances of the concepts defined in the metamodel, all relations between objects are valid with respect to relationships defined in the metamodel and if all properties are satisfied.

[1] One can notice that some properties could have been modeled with EMOF by choosing another structure for concepts and relationships. However, the number of concepts and relationships would have increased, hampering the understandability of the metamodel and increasing the distance between the metamodel and a straightforward representation of domain concepts.

Definition 1. ***Metamodel****. A metamodel is defined as the composition of:*

- ***Concepts****. The core concepts and attributes that define the domain.*
- ***Relationships****. Relationships that specify how the concepts can be bound together in a model.*
- ***Well-formedness rules****. Additional constraints that restrict the way concepts can be assembled to form a valid model.*

In this study, we consider metamodels defined with OMG standards. We distinguish two parts as defined below.

Definition 2. ***Metamodel under study****. For this work, a metamodel is defined as the composition of:*

- ***Domain structure****. An EMOF-compliant model portraying the domain concepts as metaclasses and relationships between them.*
- ***Invariants****. Well-formedness rules that impose invariants over the domain structure and that are expressed in OCL.*

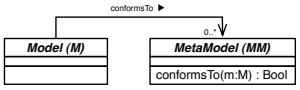

Fig. 2. Model & MetaModel Definition with Class Diagram Notation

3.2 Summary of EMOF and OCL

Figure 3 displays the structure of EMOF [2]. EMOF allows to specify the concepts of a metamodel in a `Package`. This `Package` contains `Class`es and `Property`s to model the concepts and relationships. The `Property`s of a `Class` can be either: attributes of type `Boolean`, `String` or `Natural`; or references to other `Class`es, in this case the `Property` is of type another `Class`. Figure 4 displays the structure of OCL expressions [4] that can be used to constrain the structure defined with EMOF. The most noticeable constructs for OCL expressions are: the ability to declare `Variable`s, whose type is a concept modeled with EMOF; the ability to use control structures such as `IfExp` and `LoopExp`; the ability to have composite OCL expressions, through `CallExp`s. Figure 5 displays the connection between EMOF and OCL [4, p.169]. This figure specifies that it is possible to define `Constraint`s on `Element`s (everything in EMOF is an `Element`, cf. figure 3). They can be defined as `Expression`s, and one particular type of expression is `ExpressionInOCL`, an expression defined with OCL. The most important concept is the notion of `ExpressionInOCL` that binds an `Element` coming from an EMOF model on one hand to an `OclExpression` on the other hand. The existence of this binding between formalisms is essential for metamodeling: this is how two different formalisms can be smoothly integrated in the construction of a metamodel. This binding is also what allows us to automatically analyze metamodels built with EMOF and OCL.

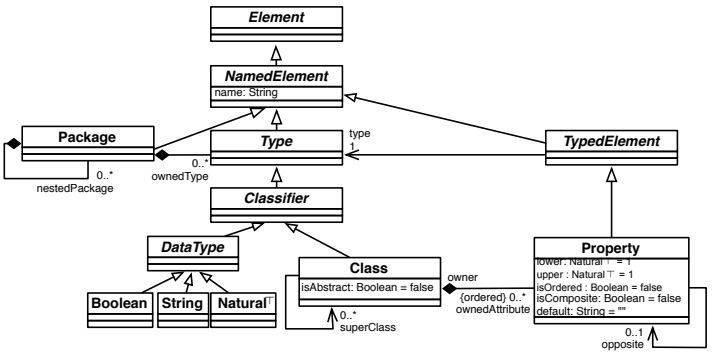

Fig. 3. The EMOF Core with Class Diagram Notation

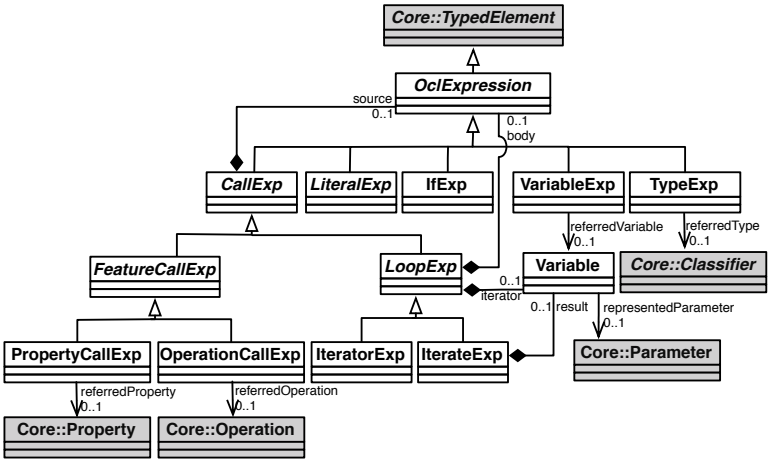

Fig. 4. OCL Expression metamodel

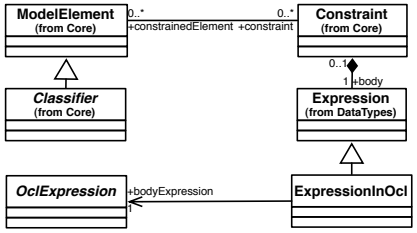

Fig. 5. OCL and MOF binding

3.3 Metrics

In order to understand the conjunct usage of these two standards, we aim at defining the following metrics.

- Size of the metamodel: The number of constructs that a metamodel provides can change dramatically from one language to the other. Such measure has to be compared when evaluating several metamodels from diverse domains.
- Size of the invariants set: Metamodels can portray different levels of complexity; highly complex domains require a large number of OCL formulae to express their logic, whereas lesser complex domains will express their knowledge with a lower number of constraints. With this metric we aim at understanding the different levels of such complexity.
- Invariant complexity with respect to the underlying domain structure: Some metamodels contain lengthy and complex well-formedness rules, while others seem to define them using simple expressions. We measure how many EMOF elements are used in each OCL invariant and thus the quantity of model elements involved in a constraint.
- Invariant complexity with respect to the OCL syntax metamodel: In order to extract the effective subset of the OCL language that is used in DSML development, we intend to query the invariants for the specific OCL expression types they use.

4 Analysis of MOF and OCL in metamodels

The data sets and metrics specified in the previous section are used to build a tool to perform the computations which will provide the data to perform analysis on a metamodel.

4.1 The Global Process for Analysis Automation

Figure 6 shows the overall process to analyze a metamodel. The process is composed of three activities with their own tools:

1. If the OCL invariants are not defined in the OCL/XMI format (extension `.oclxmi` in figure 6), the first activity consists in preprocessing (activity `OCL Parsing`). It is performed depending on the input format of the OCL invariants (extension `.ocl` in figure 6). We have used OclInEcore[2], a textual editor for Ecore files.
2. The next step consists in using an in-house built tool to automatically compute the metrics over the metamodel (activity `Metrics Computation`). Such tool would take as an input the metamodel composed of the domain structure expressed in Ecore, and the invariants expressed in OCL and produce a CSV file containing all the metric values for the input metamodel.

[2] OclInEcore, cf. `http://wiki.eclipse.org/MDT/OCLinEcore/`.

3. The metric values are finally analyzed with R[3] (activity `Statistical Analysis`). R is an open-source language for statistical applications, which provides several functionalities to run analysis and create plots, both one-variable and multi-variable. We provide a set of generic scripts that could be used for any CSV file produced with our in-house built tool. These scripts automate the production of graphics to assist analysis.

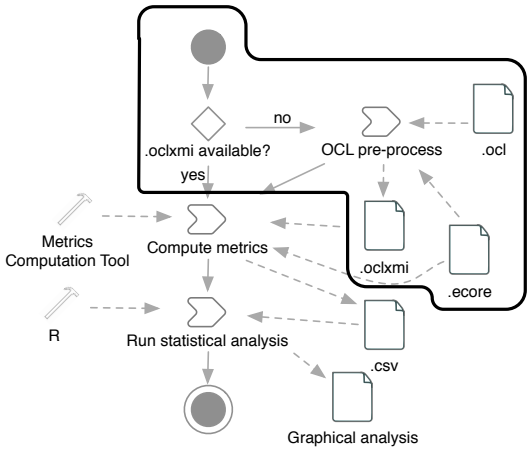

Fig. 6. SPEM Process for Metamodel Automatic Analysis

Currently our research has accomplished the encircled parts of the diagram, performing the preparation of the data of the metamodels presented in the next section.

5 Data setup and preprocessing

Understanding the use of EMOF and OCL requires a sample containing data from repositories in diverse backgrounds. The sample subjects must come from standard bodies, academia and industry altogether.

5.1 Experimental data setup

Table 1 details a list of standard specifications coming from different sources, defining domain-specific languages. The first two columns contain the name and source; the first group comes from the OMG. The following group presents metamodels taken from academic research; first a metamodel for the B language created at IMAG; SAD3 is a software architecture component model created at

[3] R, cf. `http://www.r-project.org/`

ENSTA Bretagne. In the last group, metamodels MTEP and XMS are metamodels created by Thomson Video Networks for encoding standards for video hardware. SAM is a metamodel from the Topcased open source software project. The third column counts the number of metamodels. In the OMG group, specifications define large modeling languages, normally structured in packages, therefore we treat each one of these as a separate metamodel. In the remaining cases, each specification contains only one metamodel. The fourth column mentions the formalism used to express well-formedness rules. As expected, we chose specifications using OCL. The fifth column shows the different standards that exist to specify the domain structure. Of our main interest, Ecore is a lightweight implementation of EMOF [5], providing equally an XMI-based persistence mechanism. The sixth column presents the format for expressing invariants in OCL. These are found either as separate .ocl text files or embedded in .ecore as annotations. Availability of the Ecore format and some of the mentioned forms of OCL invariants are necessary to enable the automation of the metrics computation.

Table 1. Specifications containing sample metamodels.

Name	Source	Meta-models	Expression of Constraints	Domain Structure format	OCL invariants format
UML	OMG	13	Text and OCL	Ecore	Annotations in Ecore
CCM	OMG	4	Text and OCL	Ecore	Text in documentation
OCL	OMG	4	Text and OCL	Ecore	Text in documentation
MOF	OMG	2	Text and OCL	XMI	.ocl text file
CWM	OMG	21	Text and OCL	Ecore	Text in documentation
DD	OMG	3	Text and OCL	XMI	Annotations in Ecore
B language	Academic Research	1	Language specification and OCL	Ecore	.ocl text file
SAD3	Academic Research	1	OCL	Ecore	.ocl text file
MTEP	Industry	1	OCL	Ecore	.ocl text file
XMS	Industry	1	OCL	Ecore	.ocl text file
SAM	Industry	1	OCL	Ecore	.ocl text file

5.2 Preprocessing data for analysis

The preprocessing step is based on an automated Java program that takes an Ecore/XMI metamodel with associated OCL invariants stored in their available format for each metamodel and its OCL invariants, according to table 1 and produces as output an Ecore/XMI metamodel with OCL/XMI invariants, where

all the OCL invariants that remain are syntactically correct (parse without errors using the Eclipse OCL parser [1]). The OCL/XMI format presents the abstract syntax tree of each OCL expression. At the end of the preprocessing step, every metamodel can be automatically analyzed for metrics computation. The metrics computation tool will be able to compute metrics on the MOF structure and the OCL invariants, and consequently analyzable data is emitted as output. Throughout this process we have observed the following issues.

Different storage formats Ecore is the de-facto standard based on the XMI format used to express the domain structure of a metamodel, yet there is no evidence of such a format to store OCL expressions for a metamodel. Besides OCL text files, invariants are also added as annotations; however these only consist of maps of string-to-string entries, which can themselves present different schemas. Our preprocessing program automatically detects the format and proceeds to parse and produce the previously mentioned output.

Different OCL syntaxes Different parsers allow or reject certain OCL constructs [7]. To enable automation analysis of the OCL expressions, such variations must be streamlined to satisfy the precise syntax required for Eclipse OCL; this was performed by replacing the unrecognized constructs by its accepted equivalents; for example, the use of the minus "-" operator to exclude elements from a collection, instead of the *exclude* operation.

Errors in invariants In many cases, OCL invariants are added to a metamodel with the sole purpose of documentation and might not be checked for correctness. The studied sets of invariants from the selected specifications contained incorrect OCL expressions, containing errors from syntax (invalid use of OCL constructs) or semantics (references to non-existent model elements from the domain structure). Table 2 presents trivial errors and thus capable of being corrected, as well as those that could not be fixed, since it would require knowledge from the domain expert.

6 Conclusions

In this article we have illustrated several issues that arise when metamodeling with the MOF and OCL formalisms. Our purpose is to learn how these formalisms are used in existing metamodels, in order to assist metamodelers in the future. The rest of the paper discusses a set of metamodels that we have gathered from different sources (OMG, open source project, industry) and the lessons we have learned while manually analyzing and fixing these metamodels to get them ready for automatic analysis. Most of the problems to automate the analysis over the metamodels are that OCL well-formedness rules first are provided in a variety of formats and secondly are often inconsistent with the MOF domain model. The next step for this work is to build a tool that can automatically analyze metamodels. This tool should compute a set of metrics about the

Table 2. Corrected errors in OCL invariants.

Corrected errors	**Frequency**
Missing parenthesis	94
Notation for enumeration literals	51
Missing variable in forAll body	30
Missing mandatory typecast (oclAsType())	22
Typos in pointers to metaclasses and properties	15
Typos in OCL operations invocation	13
Use of '-¿' instead of '.' for non-collection properties	10
Use of '.' as a shortcut for 'collect'	9
Use of unescaped OCL keywords	6
'if' expression without 'else' and 'endif'	5
Use of 'notEmpty' and 'isEmpty' for non-collection properties instead of oclIsUndefined()	4
Treating of boolean values as literals '#true' and '#false'	3
Use of 'union' instead of 'concat' to concatenate strings	2
Errors remaining incorrect	**Frequency**
Pointers to nonexistent properties/operations	133
Invariants with a context metaclass in an outside metamodel	2
Reference to undefined stereotypes	1

conjunct use of MOF and OCL. These metrics will be the basis for our empirical investigation. Such empirical work will lead to complement existent guidelines for metamodelers [8, 9, 6, 10] suggesting an appropriate use of MOF and OCL.

References

1. Eclipse ocl, `http://www.eclipse.org/modeling/mdt/?project=ocl`
2. Omg meta object facility core, v2.0 (2006)
3. A Framework to Formalise the MDE Foundations. In: TOWERS. pp. 14–30 (2007)
4. Omg object constraint language, v2.2 (2010)
5. Budinsky, F., Merks, E., Steinberg, D.: Eclipse Modeling Framework 2.0. Addison-Wesley Professional (2009)
6. Garcia, M.: Efficient integrity checking for essential mof+ ocl in software repositories. Journal of Object Technology 7(6)
7. Gogolla, M., Kuhlmann, M., Büttner, F.: A benchmark for ocl engine accuracy, determinateness, and efficiency. In: MoDELS. LNCS, vol. 5301, pp. 446–459 (2008)
8. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design guidelines for domain specific languages. In: The 9th OOPSLA workshop on DSM (2009)
9. Kovari, P.: Explore model-driven development (mdd) and related approaches: Applying domain-specific modeling to model-driven architecture. IBM Developerworks (2007)
10. Loecher, S., Ocke, S.: A metamodel-based ocl-compiler for uml and mof. Electron. Notes Theor. Comput. Sci. 102, 43–61 (November 2004), `http://dx.doi.org/10.1016/j.entcs.2003.09.003`
11. Selic, B.: Uml 2 specification issue 6462. http://www.omg.org/issues/issue6462.txt (2003), updates dating until 2008.